Simulations on the Grid

Fun wth the OSG

I'll report on a massive simulation project that Jin and I ran on the Open Science Grid

Jin had asked for help to run his pythia simulations for fsPHENIX

Simulations of single+ muon events in the forward region

Wanted: 5000 billion events. 5,000,000,000,000.

(I'm suspecting that he had initially asked about the simulations in jest, since he didn't really think it's doable)

History

I had run “massive” projects already in the context of the medical imaging

Previous projects in the order of 15-20 CPU-Years

There we were “guests” by virtue of the connection to SBU’s Biomedical Dept, without in-kind contributions

Now we are players among equals since the RCF provides resources back to the Grid

A lot of issues

Our existing PHENIX software is totally grid-agnostic.

Not by design, but by implementation only fit to run within the RCF:

- virtually anything you do needs access to the database
- even if there's no legitimate reason to pull, say, calibrations from the DB, that dreaded RunToTime will access the DB
- an excessive library dependency tree - in first approximation, your job pulls in *everything*.
- you run simulations? Your job still loads the event libraries, and all *raw data* reconstruction libraries
- any job you run has pulled in at least 1550MB in libraries by the time it actually starts (back in April)

Two fundamental ways to go about it

In order to run your jobs on a “foreign” node that you have no control over, you can

- bring the RCF environment to the remote node, or
- tweak your setup so it is more tolerant to more “vanilla” environments

Matt Snowball from LANL has been working on the former approach.

We have an externally accessible copy of our database set up, so external jobs have now readonly database access.

The code base can come in through a file system called *cvmfs* - think of it as “afs for the Grid”. Or afs itself.

None of that was available at the time we started the project.

Other issues

I also have some philosophical issues with the depth of the (in principle ok) approach to “export” the RCF environment

- first, the goal should be to minimize the library and other dependencies.
- a “blind” export can conflict with being a good citizen on the OSG.
- it can lead to an excessive amount of “startup” data transferred *per job*.
- exhaust all possibilities to minimize the “startup costs” first
- (with sPHENIX and a fresh start, we have a chance to get this right)

What we did, and didn’t

We have access to a file system called *cvmfs*

- each new node still pulls all support libraries from that network-based filesystem
- cvmfs has a caching mechanism like afs, but each grid node needs to initially pull that over through the network (again, *exactly* like afs)
- instead, I use condor’s file transfer mechanism to ship the “support bundle” over
- I bzip2-compress that to 17% of the orginal size = 263MB
- I made a special custom version of `pdbcal` that will leave the DB alone when the run number is 0
- the jobs run 10-15 hours, so this is negligible *and* rock-solid

64bit, please!

- 64bit appears to be the binary format of the Grid
- a good fraction of nodes isn't even set up to run 32bit binaries
- (no amount of support libs will make those nodes run 32bit code)
- I converted the entire project to 64bit

Libraries

I’m shipping two areas with libraries:

- one set of “core” libraries which are needed to run the project at hand
- another of “in case they are not there” set of what are nominally system libs
- trial-and-error to identify those from the error logs in the setup phase
- you wonder why a batch-based job would want an Nividia X11 library... but that’s what it is.

What we get

8563 running jobs at that moment...

```
261664.0   purschke        4/23 00:05   0+00:00:00 I  0   0.0  run.sh 249990
261665.0   purschke        4/23 00:05   0+00:00:00 I  0   0.0  run.sh 249991
261666.0   purschke        4/23 00:05   0+00:00:00 I  0   0.0  run.sh 249992
261667.0   purschke        4/23 00:05   0+00:00:00 I  0   0.0  run.sh 249993
261668.0   purschke        4/23 00:05   0+00:00:00 I  0   0.0  run.sh 249994
261669.0   purschke        4/23 00:05   0+00:00:00 I  0   0.0  run.sh 249995
261670.0   purschke        4/23 00:05   0+00:00:00 I  0   0.0  run.sh 249996
261671.0   purschke        4/23 00:05   0+00:00:00 I  0   0.0  run.sh 249997
261672.0   purschke        4/23 00:05   0+00:00:00 I  0   0.0  run.sh 249998
261673.0   purschke        4/23 00:05   0+00:00:00 I  0   0.0  run.sh 249999
261674.0   purschke        4/23 00:05   0+00:00:00 I  0   0.0  run.sh 250000

34010 jobs; 0 completed, 0 removed, 25442 idle, 8563 running, 5 held, 0 suspended
```

How it’s done

Submit jobs in batches of 100,000

```
#! /bin/sh
AREA="/local-scratch/purschke/pythia"
for n in $(seq 500001 510000) ; do

    JOBNR=$(printf "%08d\n" $n)

    LOGFILE="$AREA/log/pythia_${JOBNR}.log"
    OUTFILE="$AREA/log/pythia_${JOBNR}.out"
    ERRFILE="$AREA/log/pythia_${JOBNR}.err"

    condor_submit condor_pythia -a "output = $OUTFILE" \
          -a "error = $ERRFILE" \
          -a "Log = $LOGFILE" \
          -a "Arguments = $n"
done
```

Condor

`condor_pythia` looks like

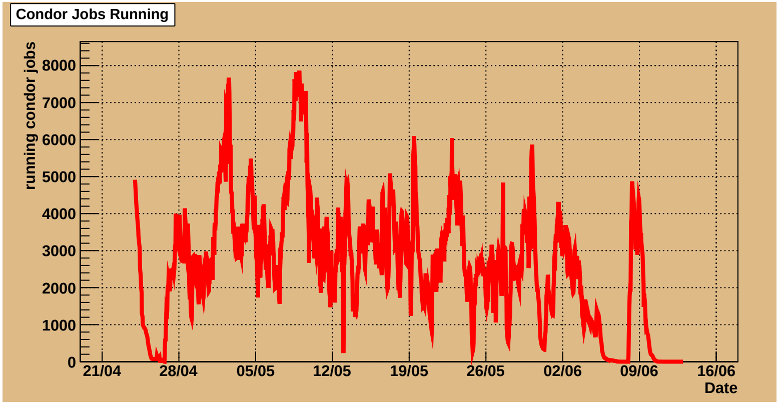
```
InitialDir = /local-scratch/purschke/pythia/output
Executable      =  run.sh
Universe = vanilla
when_to_transfer_output = ON_EXIT
notification= Never

should_transfer_files = YES

transfer_input_files = ../files.tar.bz2

output = x.out
error = x.err
Log = x.log

requirements = (( OpSysAndVer == "rhel6" ) ||  ( OpSysAndVer == "SL6" ) ||
+ProjectName = "sPHENIX"
queue
```

files.tar.bz2

- The `files.tar.bz2` has the “environment” the job needs
- I use the condor file transfer mechanism to furnish that file
- The master script unpacks it in the condor scratch area, sets up `LD_LIBRARY_PATH` and `PATH`
- at the end, the script deletes everything but the output files, transfers those back
- done

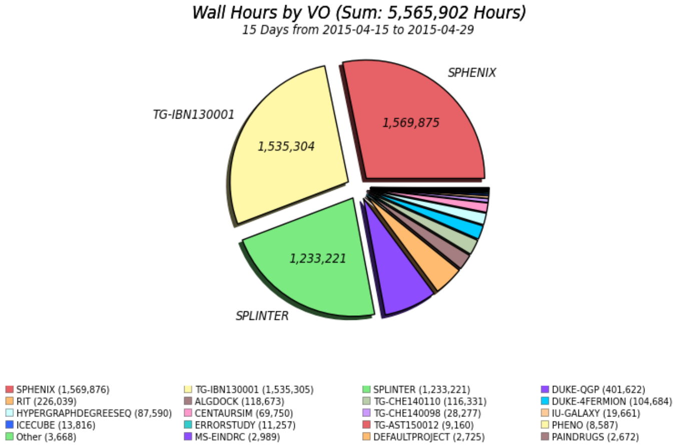
Jobs Running

I began gathering job ststistics around the halfway-point of the project



The dip in April is due to a major maintenance in the central hosts

We did get noticed by the powers of the OSG...



This is our CPU usage on the OSG by the end of April.

Performance

How many hours/days/years have we run?

In the 2nd half of April:

```
$ find log/ -name ``*.log'' -exec grep 'Total Remote Usage' {} \; | \
   sed -e 's/,//g' | awk '{print $3}' | \
   awk -F: '{X += ($1 *3600 + $2*60 + $3)/3600} END {print X}'

1.34523e06

$ bc -l
bc 1.06

1345230 / 24
56051.25000000000000000000
1345230 / 24/365
153.56506849315068493150
```

153 years in April...

We used about **289 years of CPU time** total in the end. The project was completed around June 10.

Operations

- After the initial setup (requiring some trial-and-error), all ran on autopilot
- Some tuning to find the sweet spot for job succes rate and minimal I/O overhead
- I checked about twice a week for issues and to submit new jobs as needed
- Less than 0.04% job failure rate
- very minor time committment needed once in steady-state

Summary

- We have the resources to run simulations at a scale that was unthinkable maybe 2 years ago
- I was surprised by the extent to which we ran through open doors with the OSG management
- We didn't ask for much in terms of support though
- Most important: **be a good citizen!** It's a shared resource. I got a lot of kudos for being conscientious.
- Not all projects lend themselves to be run on the grid - a wrong IO/CPU time ratio leads to waste
- finally - 64 bit, please. Nodes set up run 32bit are the minority.
- Next: between Matt and myself, maybe find some middle ground between exporting the RCF environment and "gridifying" the project, in a semi-automated fashion

Final Words

- Do not self-censor yourself by assuming a project cannot be done
- There is no guantee that it can
- The landscape of what is possible has changed quite a bit though.